~~by the user and conduct such a search when requested for all instances that satisfy the user-specified search criteria;~~

~~writing code to determine automatically which services of an object can be invoked given the current state of the object and only allow those services to be invoked;~~

~~writing code to furnish initial values for object-valued arguments of services and receive any user input arguments;~~

~~writing code to check data type entered by a user for validity for the argument the data fills and make sure the entered data is within a valid range for the argument the data is intended to fill;~~

~~writing code to check for dependencies between arguments, and, if a dependency exists, and user input data triggers the dependency, to display an appropriate dialog box prompting the user to enter input data needed to satisfy the dependency;~~

~~writing code to invoke the appropriate object server code linked to a particular service when a user makes an input indicating a desire to invoke that service and to pass the object server code the appropriate arguments for the service;~~

~~writing code to wait for results of execution of a service, and to display an error message if an error occurred, but, if no error occurred, to wait for further user input.~~

REMARKS

Claim 1 Argument

Claim 1 was rejected under 35 USC 102 as anticipated by Bruegge, "Rose Tutorial" describing the Rational Rose CASE tool. In response to this rejection, claim 1

has been amended to specify that the computer is programmed to provide a graphical user interface which includes tools through which a user can enter primitive data which defines an object model, a functional model, a dynamic model and a presentation model, all of which, taken together, define a complete conceptual model of a program for which software code is to be automatically written. The amendments further specifiies that the computer is also programmed to then convert each piece of data entered by the user to a corresponding part of a formal language specification which defines the user's requirements for the computer program to be written and that each part of the formal language specification is expressed in one or more statements in a formal language which has a syntax and semantics which are understood by a validator process.

The difference between the invention of claim 1 and Bruegge is that Bruegge uses a graphical user interface to define ONLY an object model of the program and then converts that object model directly to Java code, apparently without any validation step between creation of the model and generation of the code. The fact that the Rational Rose tool graphic user interface can only be used to create an object model is seen from page 5 of the Bruegge "Rose Tutorial". There it is stated:

> Rose enables the user to:
>
> * construct an object model of the system
> * draw diagrams representing different views of the model
> * generate documents describing the model
> * generate Java code

There is no mention that Rose enables a user to use tool of the Rational Rose the graphic user interface to enter primitives which define a functional model or a dynamic model or a presentation model. Support in our specification for these amendments to the claims at bar is found in great detail in Appendix A

below. Because the Bruegge reference describes a Rational Rose tool which cannot create functional models, dynamic models or presentation models, there are no tools in the Rational Rose graphical user interface which are presented to the Rational Rose user to create such models. Therefore, claim 1 is not anticipated by a computer programmed with the Rational Rose program.

The functional model, dynamic model and presentation model are necessary to automatically generate a complete computer program. The functional model includes primitives entered by the user through the GUI tools to specify valuations, i.e., mathematical or Boolean equations which specify the effect specified events have on the values of variable attributes of an object. The dynamic model allows a user to enter primitives that define state transition diagrams and object interaction diagrams. The state transition diagram contains primitives entered by a user to specify event preconditions (formulas labeling the event transitions) and the process definition of a class where the template for valid object lives is fixed. From the object interaction diagram of the dynamic model, two other features of an OASIS class specification are included: trigger relationships and global transactions. (see page 40, line 4 of our specification for support for the above). The presentation model is comprised of primitives entered by the user through the GUI tools to specify pre-defined concepts that are used to describe the user interface requisites of the program to be automatically generated. User interfaces tend to fall into patterns, and the presentation model specifies which pattern of user interface is to be used and the content to be displayed in that pattern.

In contrast, the Rose tool can only be used to create an object model of the program to be written so its GUI does not present any dialog boxes or menu choice to allow a user to enter valuation formulas and events and attribute names for a functional model, or state transition diagrams or object interaction diagrams of a dynamic model nor

the desired user interface pattern and specification of its content to define a presentation model.

Unlike the invention of claim 1, as amended, in Bruegge, there is no conversion of the object model into a formal specification as an intermediate step prior to generation of Java code.

This is a key difference because it is the conversion of the conceptual model in the invention of claim 1, as amended, into the formal language specification which allows the validation of the formal specification and it is this validation which creates error free code. In contrast, the Bruegge reference goes directly from the object model the user creates to code. If the user made an error in defining the object model, such as giving a variable two different definitions or defining a variable data type in an inconsistent manner such a floating point where the service was expecting to output a string.

Claim 2 was rejected as anticipated by the Bruegge reference. In response, claim 2 was amended to clarify that the process includes steps of displaying on a computer tools which include dialog boxes, menu choices and/or graphic screens which can be invoked to allow a user to enter primitives which define "an object model, functional model, dynamic model and presentation model, which, taken together, define a full conceptual model of a computer program to be written by an automatic software production tool". A step was also added of converting the primitives so entered into statements in a mathematically based formal language which has predefined rules of syntax and semantics.

Claim 3 was amended voluntarily to change what it claims sinces its original subject matter was added to its parent claim 2. Claim 3, as amended, claims a step of validating the statements in the formal language which were created from the primitives the user entered so as to verify that the statements are syntactically and semantically

correct, complete and not ambiguous. Since claim 3 depends from claim 2 which has been amended to specify process steps of displaying user interface tools which the Brugge prior art user interface does not display, claim 3 is also not anticipated.

Claim 4 was rejected as anticipated. In response it was amended to add the fact that the computer-executable instructions control the computer to display tools by which a user can enter primitives which define an object model, a functional model, a dynamic model and a presentation model. The Bruegge reference does not teach such a medium programmed with these types of computer instructions.

Claim 5 depends from claim 4, and is therefore not anticipated for the same reason claim 4 is not anticipated. A few minor amendments were made voluntarily to improve the form of the claim.

Appendix A – Distinctive Primitives Which Can Be Specified in a Conceptual Model Using the Tool of the Claimed Invention Which Cannot Be Specified By The Prior Art Tools

Local Transactions (Object Model)

Purpose

A transaction is a molecular service: that is, a service whose functionality is defined by a sequence of other services. The services composing a local transaction can be atomic (events) or molecular (transactions).

The combination of the valuations defined in the Functional Model (which determine the functionality of events) with the composition mechanism of transactions result in the ability to completely define the functionality of services (either atomic or molecular).

Supporting Figure(s) in our specification

Figure 13 **IN CHG-001.1P and CHG-001.2P CIPs**

Supporting Passages in our specification

Page 63, line 5

"FIG. 13 is a screenshot of the dialog box used to create one formula in a local transaction carried out by a composed service (single services are called events, and composed services are called local transactions)."

Page 20, line 3

"Services can be of two types: events and transactions. Events are atomic operations while transactions are composed of services which can be in turn events or transactions. Every service can have the following characteristics: name, type of

service (event or transaction), service alias, remarks and a help message. Events can be of three types: new, destroy or none of them. Events can also be shared by several classes of the project. Shared events belong to all classes sharing them. Transactions have a formula that expresses the composition of services."

Page 27, line 3

"An "event" as used in the claims means a single service and not a transaction which is defined as a composed or complex service (which means more than one service executes)."

Page 39, line 1

"The molecular units that are the transactions have two main properties. First, they follow an all-or-nothing policy with respect to the execution of the involved events: when a failure happens during a transaction execution, the resultant state will be the initial one. Second, they exhibit the non-observability of intermediate states."

Valuations (Functional Model)

Purpose

The functional model defines how the occurrence of events of a class change the values of attributes of said class. This is done by means of valuations.

Figure(s) in our specification

Figure 5

Figure 15 **IN CHG-001.1P and CHG-001.2P CIPs**

Passages in our specification

Page 12, line 12

"FIG. 5 illustrates an exemplary dialog for receiving input for the functional model."

Page 13, line 10

"FIG. 15 is a dialog box to enter the functional model formulas that define evaluation of the attribute "cause" with the "modify" event (an event is a single service). The functional model relates services mathematically through well-formed formulas to the values of attributes these services act upon."

Page 17, line 13

"In one implementation, the Conceptual Model is subdivided into four complementary models: an object model, a dynamic model, a functional model, and a presentation model."

Page 18, line 19

"Rather, four complementary models, that of the object model, the dynamic model, the functional model and the presentation model, are employed to allow a designer to specify the system requirements."

Page 19, line 7

"In accordance with the widely accepted object oriented conceptual modeling principles, the Conceptual Model is subdivided into an object model, a dynamic model, and a functional model. These three models, however, are insufficient by themselves to specific a complete application, because a complete application also requires a user interface."

Page 26, line 29

"One embodiment of the present invention, however, employs a functional model that is quite different with respect to these conventional approaches. In this functional model, the semantics associated with any change of an object state is captured as a consequence of an event occurrence. To do this, the following information is declaratively specified: how every event changes the object state depending on the arguments of the involved event, and the object's current state. This is called. "valuation."

In particular, the functional model employs the concept of the categorization of valuations. Three types of valuations are defined.:push-pop, state-independent and discrete-domain based. Each type fixes the pattern of information required to define its functionality. Push-pop valuations are those whose relevant events increase or decrease the value of the attribute by a given quantity, or reset the attribute to a certain value. State-independent valuations give a new value to the attribute involved independently of the previous attribute's value.
Discrete-domain valuations give a value to the attributes from a limited domain based on the attribute's previous value. The different values of this domain model the valid situations that are possible for the attribute."

Page 26, line 29

"One embodiment of the present invention, however, employs a functional model that is quite different with respect to these conventional approaches. In this functional model, the semantics associated with any change of an object state is captured as a consequence of an event occurrence. Basically, the functional model allows a SOSY

modeler to specify a class, an attribute of that class and an event of that class and then define a mathematical or logical formula that defines how the attribute's value will be changed when this event happens. An "event" as used in the claims means a single service and not a transaction which is defined as a composed or complex service (which means more than one service executes). In the preferred embodiment, condition-action pair is specified for each valuation. The condition is a single math or logic formula is specified which specifies a condition which results in a value or logical value which can be mapped to only one of two possible values: true or false. The action is a single math or logical formula which specifies how the value of the attribute is changed if the service is executed and the condition is true. In other embodiments, only a single formula that specifies the change to the attribute if the service is executed is required.

The functional model is built in the preferred embodiment by presenting a dialog box that allows the user to choose a class, an attribute of that class and a service of that class and then fill in one or more formula or logical expressions (condition-action or only action) which controls how the value of that attribute will be changed when the service is executed. The important thing about this is that the user be allowed to specify the mathematical or logical operation which will be performed to change the value of the attribute when the service is executed, and it is not critical how the user interface is implemented. Any means to allow a user to specify the class, the attribute of that class and the service of that class and then fill in a mathematical or logical expression which controls what happens to the specified attribute when the service is executed will suffice to practice the invention. Every one of these mathematical expressions is referred to as a valuation. Every valuation has to have a condition and action pair in the preferred embodiment, but in other species, only an action need be specified. The

condition can be any well formed formula resulting in a Boolean value which can be mapped to only one of two possible conditions: true or false. The action specified in the pair is any other well-formed mathematical and/or logical formula resulting in a new value for the variable attribute, said new value being of the attribute's same data type (type of data of action must be compatible with the type of data of the attribute). This valuation formula can be only mathematical or only a Boolean logical expression or a combination of both mathematical operators and Boolean logical expressions.

Regardless of the user interface used to gather data from the user to define the valuations in the functional model, all species within the genus of the invention of generating functional models will generate a data structure having the following content: data defining the valuation formula which affects the value of each variable attribute (the data that defines the valuation formula identifies the service and the attribute affected and the mathematical and/or logical operations to be performed and any operands needed). This data structure can be any format, but it must contain at least the above identified content.

To define the functional model, the following information is declaratively specified by the SOSY modeler: how every event changes the object state depending on the arguments of the involved event, and the object's current state. This is called "valuation".

In particular, the functional model employs the concept of the categorization of valuations. Three types of valuations are defined.:push-pop, state-independent and discrete-domain based. Each type fixes the pattern of information required to define its functionality.

Push-pop valuations are those whose relevant events increase or decrease the value of the attribute by a given quantity, or reset the attribute to a certain value.

State-independent valuations give a new value to the attribute involved independently of the previous attribute's value.

Discrete-domain valuations give a value to the attributes from a limited domain based on the attribute's previous value. The different values of this domain model the valid situations that are possible for the attribute."

Page 29, line 1

"This categorization of the valuations is a contribution of one aspect of the present invention that allows a complete formal specification to be generated in an automated way, completely capturing a event's functionality.
Accordingly, the functional model is responsible for capturing the semantics of every change of state for the attributes of a class. It has no graphical diagram. Textual information is collected through an interactive dialog that fills the corresponding part of the Information Structures explained before. FIG. 5 illustrates an exemplary dialog for receiving input for the functional model."

Page 37, line 12

"Evaluations are formulas of the form _[a] _ whose semantics is given by defining a r function that, from a ground action a returns a function between possible worlds. In other words, being a possible world for an object any valid state, the r function determines which transitions between object states are valid after the execution of an

action a. In the example,there are the following evaluations:

[loan( )] book_count=book_count+1;

[returns( )] book_count=book_count-1;

Within this dynamic logic environment, the formula F is evaluated in s Œ W, and F' is evaluated in r(a), with r(a) being the world represented by the object state after the execution in s of the action considered."

Page 40, line 12

"Finally, the functional model yields the dynamic formulas related to evaluations, where the effect of events on attributes is specified."

Page 63, line 16

"Phase 4: Express evaluations. During this phase, one or more dialog boxes are presented to the SOSY modeler wherein he or she expresses evaluations of what will be the effect of all event for each variable attributes of each class.

This is the process of building the functional model portion of the Conceptual Model. The value change of an attribute when an event happens is known as "evaluation".

FIG. 15 is a dialog box to enter the functional model formulas that define evaluation of the attribute "cause" with the "modify" event (an event is a single service). The functional model relates services mathematically through well-formed formulas to the values of

attributes these services act upon. Note that at box 724, the SOSY modeler has not filled in an evaluation formula that could be encoded in the final code to do a calculation to change the value of "cause" when the modify event occurs. Instead, as seen from box 726, the value of "cause" will be changed to whatever the value of the argument "p_cause" of the event "modify" when "modify" is executed."

Dynamic Model

Purpose

The dynamic model specifies the behaviour of an object in response to services, triggers and global transactions. It encompasses:

- The State Transition Diagram
- The Object Interaction Diagram

Figure(s) in our specification

Figure 4A (State Transition Diagram)

Figure 4B (Object Interaction Diagram)

Figure 17 (State Transition Diagram) **IN CHG-001.1P and CHG-001.2P CIPs**

Passages in our specification

Page 13, line 17

"FIG. 17 is one of the two graphical user interface diagrams of the dynamic model on which the SOSY modeler has drawn a graphic illustrating the state transitions for the "expense" class."

Page 24, line 20

"The dynamic model specifies the behavior of an object in response to services, triggers

and global transactions. In one embodiment, the dynamic model is represented by two diagrams, a state transition diagram and an object interaction diagram.

The state transition diagram (STD) is used to describe correct behavior by establishing valid object life cycles for every class. A valid life refers to an appropriate sequence of states that characterizes the correct behavior of the objects that belong to a specific class. Transitions represent valid changes of state. A transition has an action and, optionally, a control condition or guard. An action is composed of a service plus a subset of its valid agents defined in the Object Model. If all of them are marked, the transition is labeled with an asterisk (*). Control conditions are well formed formulas defined on object attributes and/or service arguments to avoid the possible non-determinism for a given action. Actions might have one precondition that must be satisfied in order to accept its execution. A blank circle represents the state previous to existence of the object. Transitions that have this state as source must be composed of creation actions. Similarly, a bull's eye represent the state after destruction of the object. Transitions having this state as destination must be composed of destruction actions. Intermediate states are represented by circles labeled with an state name.

Accordingly, the state transition diagram shows a graphical representation of the various states of an object and transitions between the states."

Page 25, line 15

"Transitions are represented by solid arrows from a source state to a destination state. The middle of the transition arrow is labeled with a text displaying the action, precondition and guards (if proceeds)."

Page 25, line 24

"The object interaction diagram specifies interobject communication. Two basic interactions are defined: triggers, which are object services that are automatically activated when a pre-specified condition is satisfied, and global transactions, which are themselves services involving services of different objects and or other global transactions. There is one state transition diagram for every class, but only one object interaction diagram for the whole Conceptual Model, where the previous interactions will be graphically specified.

In one embodiment, boxes labeled with an underlined name represent class objects.Trigger specifications follow this syntax:destination::acti- on if trigger-condition. The first component of the trigger is the destination, i.e., the object(s) to which the triggered service is addressed. The trigger destination can be the same object where the condition is satisfied (i.e. self), a specific object, or an entire class population if broadcasting the service. Finally, the triggered service and its corresponding triggering relationship are declared. Global Transactions are graphically specified by connecting the actions involved in the declared interaction. These actions are represented as solid lines linking the objects (boxes) that provide them.

Accordingly, communication between objects and activity rules are described in the object interaction diagram, which presents graphical boxes, graphical triggers, and graphical interactions. FIG. 4B illustrates an exemplary object interaction diagram 420 in accordance with one embodiment of the present invention.

In the object interaction diagram 420, the graphical interactions is represented by lines for the components of a graphical interaction. Graphical boxes, such as reader box 422, are declared, in this case, as special boxes that can reference objects (particular or generic) such as a reader. Graphical triggers are depicted using solid lines that have a text

displaying the service to execute and the triggering condition. Components of graphical interactions also use solid lines. Each one has a text displaying a number of the interaction, and the action that will be executed. In the example, trigger 424 indicates that the reader punish action is to be invoke invoked when the number of books that a reader is currently borrowing reaches 10."

Page 40, line 5

"The dynamic model uses two kind of diagrams, the state transition diagram and the object interaction diagram. From the state transition diagram, the following are obtained: event preconditions, which are those formulas labeling the event transitions: the process definition of a class, where the template for valid object lives is fixed. From the object interaction diagram, two other features of an OASIS class specification are completed: trigger relationships and global transactions, which are those involving different objects."

Page 64, line 17

"FIG. 17 is one of the two graphical user interface diagrams of the dynamic model on which the SOSY modeler has drawn a graphic illustrating the state transitions for the "expense" class. Each state in the state transition diagram represents a valid state for the object and represents one of the "valid lives" and really is one of the unseen attributes of the expense class. An object can only enter one of the displayed states if the corresponding service has been thrown to transition to it from a previous state."

[Col. 9, lines 28-32]

"A class can also store triggers. Each trigger may be composed of trigger target specified in terms of self, class or object, trigger condition, triggered action (service plus

a list of possible agents) to be activated and a list of default values associated with the arguments of the related service."

Page 38, line 8

"Triggers are formulas of the form _[_a]false,, where ¬a is the action negation. This formula means that a does not occur, and what does occur is not specified. If F holds and an action other than a occurs, then there is no successor state. This forces a to occur or the system remains in a blocked state. For instance, using the appropriate dynamic formula where we include in the triggered service information about the destination (according to the trigger expressiveness presented when the object interaction diagram 420 was introduced), we will declare:

book_count=10 [Self::punish( )] false

This trigger may be written in an equivalent but more conventional way for specification purposes as:

Self::punish( ) if book_count=10;

Thus, triggers are actions activated when the condition stated in F holds. The main difference between preconditions and triggers comes from the fact that in triggers there is an obligation to activate an action as soon as the given condition is satisfied. In this way triggers allow us to introduce internal activity in the Object Society that is being modeled."

Presentation Model

Purpose

Describe user interface requisites in an abstract way, regardless of how these requisites are to be implemented.

Figure(s) in our specification

FIG. 19 (Display Set) **IN CHG-001.1P and CHG-001.2P CIP**

FIG. 20 (Filter) **IN CHG-001.1P and CHG-001.2P CIP**

Passages in our specification

Page 13, line 20

"FIG. 18 is a dialog box used by the SOSY modeler to establish this precondition."

Page 13, line 21

"FIG. 19 is a dialog box used by the SOSY modeler to establish the set of attributes which will be displayed for the "expense" class."

Page 11, line 16

"Another aspect of the present invention stems from the realization that a major source of inadequacy of conventional prototyping techniques is that these techniques lack the capability to specify the user interface aspects. Thus, such conventional prototypes have primitive user interfaces that are unacceptable for final, customer-ready software application. Accordingly, this aspect of the invention relates to an automated software production tool, software, and methodology that includes a formal specification of a Conceptual Model that specifies requirements for a software application. The Conceptual Model includes a presentation model that specifies patterns for a user interface of the software application. The formal specification, which also specifies the presentation

model is validated; and the software application is then generated based on the validated formal specification. As a result, the generated software application includes instructions for handling the user interface in accordance with the patterns specified in the presentation model."

Page 29, line 11

"The presentation model is a set of pre-defined concepts that can be used to describe user interface requisites. These concepts arise from distilling and abstracting repetitive scenarios in developing the user interfaces. These abstractions of the repetitive scenarios are called patterns. A set of patterns is called a pattern language.

In this sense, the presentation model is a collection of patterns designed to reflect user interfaces requirements. A pattern is a clear description of a recurrent problem with a recurrent solution in a given restricted domain and giving an initial context. The documented patterns abstract the essence of the problem and the essence of the solution and therefore can be applied several times to resolve problems that match with the initial context and domain. The pattern language is composed of a plurality of patterns. The present invention is not limited to any particular list of patterns, but the following is a brief description of some user interface patterns that have been found to be useful: Service presentation pattern, Instance presentation pattern, class population presentation pattern, master-detail presentation pattern and action Selection presentation pattern."

Primitive: Service Presentation Pattern

(currently referred to as Service Interaction Unit)

Page 29, line 25

"A service presentation pattern captures how a service will enquire data to the final user. This patterns controls the filling out of service arguments and contains actions to launch the service or to exit performing no action. It is based on other lower level patterns that refer to more specific interface tasks such as an introduction pattern, defined selection pattern, population selection pattern, dependency pattern, status recovery pattern, supplementary information pattern and argument grouping presentation"

Primitive: Introduction Pattern

Page 29, line 29

"The introduction pattern that handles with restrictions to input data that must be provided to the system by the final user (i.e., the user who employs the final application). In particular, edit-masks and range-values are introduced, constraining the values that can validly be input in the interface. In this manner, the user-entry errors are reduced. This pattern can be applied to arguments in services or to attributes in classes to improve data input process through validating input arguments."

Primitive: Defined Selection Pattern

Page 30, line 3

"The defined selection pattern that specifies a set of valid values for an argument. When the input data items are static, are a few, and are well known, the designer can declare by enumeration a set containing such valid values. This pattern is similar to those that define an enumerated type and an optional default value. Accordingly, the final user can only select an entry from the pre-specified set, thereby reducing error prone input. For example, one representation of this pattern could be a Combo-Box. This pattern can be

applied to arguments in services or to attributes in classes to improve data input process."

Primitive: Population Selection Pattern

(currently part of the Population Interaction Unit)

Page 30, line 11

"The population selection pattern that handles the display and selection of objects in a multiple objects society. Specifically, this pattern contains a filter, a display set, and an order criterion, which respectively determine how objects are filtered (Filter Expression), what data is displayed (Display Set), and how objects are ordered (Order Criteria). This pattern may be thought of as a SQL Select statement with columns, where for the filter expression and order by for the ordering clauses, and can be applied to object-valuated arguments in services whenever it is possible to select an object from a given population of existing objects."

Primitive: Dependency Pattern

Page 30, line 19

"The dependency pattern, that is a set of Event-Condition-Action (ECA) rules allowing the specification of dependency rules between arguments in services. When arguments are dependent on others, these constraints use this kind of rules."

Primitive: Status Recovery Pattern

Page 84, line 17

"The status recovery pattern is an implicitly created pattern that recovers data from object attributes to initialize service arguments. This can be modeled as an implicit set of

dependency patterns."

Primitive: Supplementary Information Pattern

[Col. 15, lines 56-59]

"The supplementary information pattern handles the feedback that is provided to final users in order to assure they choose or input the correct OID (object identifier) for an existent object."

[Col. 15, lines 63-64]

"The supplementary information pattern is applicable to object-valuated arguments."

Primitive: Argument Grouping Presentation Pattern

[Col. 15, lines 65-67]

"The argument grouping presentation pattern, that captures how to group the requested service arguments according to the user wishes."

Primitive: Instance Presentation Pattern

(currently referred to as Instance Interaction Unit)

[Col. 16, lines 1-5]

"An instance presentation pattern captures how the properties of an object are presented to the final user. In this context, the user will be able to launch services or to navigate to other related objects. The instance presentation pattern is a detailed view of an instance."

Primitive: Class Population Presentation Pattern

(currently referred to as Population Interaction Unit)

[Col. 16, lines 6-10]

"A class population presentation pattern captures how the properties of multiple objects of one class are presented to the final user. In this context, once an object is selected, the final user will be able to launch a service or to navigate to other related objects. The objects can also be filtered."

Primitive: Master-Detail Presentation Patter

(currently referred to as Master-Detail Interaction Unit)

[Col. 16, lines 11-20]

"A master-detail presentation pattern captures how to present a certain object of a class with other related objects that may complete the full detail of the object. To build this pattern the following patterns are used: instance presentation, class population presentation and, recursively, master-detail presentation. In this manner, multi-detail (multiple details) and multi-level master-detail (multiple levels recursively) can be modeled. For example, one scenario involves an invoice header followed by a set of invoice lines related to the invoice."

Primitive: Action Selection Pattern

[Col. 16, lines 21-28]

"An action selection pattern captures how the services are offered to final users following the principle of gradual approach. This pattern allows, for example, generating menus of application using a tree structure. The final tree structure will be obtained from the set of services specified in the classes of the Conceptual Model. The user could launch services or queries (observations) defined in the Conceptual Model."

Primitive: Filter Expression

[Col. 16, lines 29-40]

"A Filter Expression is a well-formed formula that evaluates to a Boolean type. This formula is interpreted as follows: the objects that satisfy the formula pass the filter; the ones that do not fulfill the condition do not pass the filter. Consequently, the filter acts like a sift that only allows objects that fulfill the formula to pass. These formulas can contain parameters that are resolved at execution time, providing values for the variables or asking them directly to the final user. A filter pattern may be thought of as an abstraction of a SQL where clause, and is applied in a population selection pattern."

Primitive: Display Set

[Col. 16, lines 41-44]

"A Display Set is an ordered set of attributes that is shown to reflect the status of an object. A Display Set may be thought of as an abstraction of the columns in a SQL clause, and is applied in a population selection pattern."

Primitive: Order Criterion

[Col. 16, lines 45-50]

"The Order Criterion is an ordered set of tuples that contain: an attribute and an order (ascending/descending). This set of tuples fixes an order criterion over the filtered objects. An order criterion pattern may be thought of as an abstraction of an order by SQL clause, and is applied in a population selection pattern."

Respectfully submitted,

Dated: January 12, 2001

Ronald Craig Fish
Reg. No. 28,843
Tel 408 866 4777

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail, postage prepaid, in an envelope addressed to: Commissioner for Patents , P.O. Box 1450, Alexandria, Va. 22313-1450.
on 1/12/05
(Date of Deposit)

Ronald Craig Fish, President
Ronald Craig Fish, a Law Corporation
Reg. No. 28,843